SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 8, 35–48 (1996)

Practice

Regression Control Charts to Manage Software Maintenance

DWIGHT A. HAWORTH
College of Business Administration, University of Nebraska at Omaha, Omaha, NE 68182–0048, U.S.A.

SUMMARY

The motivation for applying control chart techniques to software processes is briefly discussed. The shortcomings of a multiple regression control chart are reviewed and a visual-programming solution application is demonstrated. In it, a multiple regression model for software maintenance is used based on general metrics and measures locally obtainable from available computer programs. The construction of the application solution is discussed in detail. The resulting solution application is not an estimating program. Rather, it is an easy-to-use analysis program. It can be used to evaluate completed software maintenance work, and to collect data for maintenance management and for longitudinal studies. It makes a powerful quality control tool available to managers of complex software maintenance processes that are appropriately modelled with multiple regression.

KEY WORDS: software maintenance; regression models; software metrics; control charts; software management

1. BACKGROUND

1.1. Organization

A medium-sized manufacturing firm with a nation-wide distribution organization is seeking a method of monitoring its software maintenance process. In the past decade, its computing environment grew from distributed minicomputers to what is now a mainframe-and-distributed-mini environment. The firm has just finished upgrading to the latest versions of its vendor's hardware.

The firm has a small data processing organization with a programming staff of ten and a programming supervisor. The code library contains approximately 500 000 executable lines of COBOL in 200 separately compiled units. The firm employs configuration management and a father–son archiving scheme. Testing is accomplished using centralized test data followed by 'beta testing' at one of the distributed locations. Successful operation at the test location triggers distribution of executable units to the remaining locations.

Maintainability has become a strategic issue for the firm. Critical customer accounts are served by customized order-entry software. The ability to provide this service is a central element of the firm's marketing strategy, and therefore, the ability to respond quickly to

CCC 1040–550X/96/010035–14

Received 6 April 1993
Revised 26 October 1995

requests from strategic customers makes software maintenance more than a cost factor. Software maintenance is now a strategic issue, and the firm is seeking methods to monitor its maintenance process and identify areas for improvement.

A proven technique for monitoring processes is the control chart, a mainstay in production management. Control charts have been ignored in software process analysis because of the unique nature of software products. However, the increasing interest in software processes and their management suggests that the application of control charts be re-examined.

1.2. Tools

The standard control chart has been described by many authors; Ryan (1989) is a recent publication. The utility of the control chart lies in its simplicity. The user enters the chart with observed values of interest and can see immediately whether the current observation lies within control limits. Moreover, the plotting of observations in sequence permits longitudinal studies of the data. Longitudinal studies allow the user to determine if a systematic deviation is occurring even though all values fall within limits. The chart in Figure 1 is an example that may be interpreted as showing an increasing deviation over time, assuming the entries are made in the order of occurrence. One interpretation may be that something changed about the time of observation 10 that caused the succeeding observations to be consistently

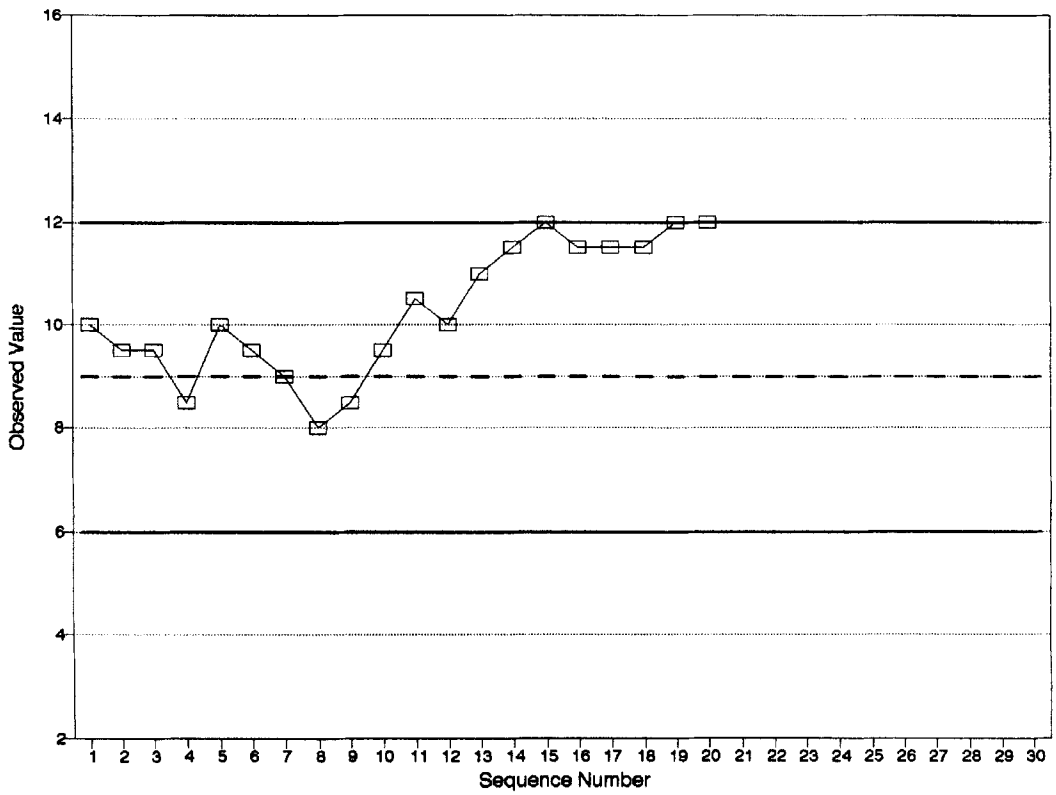

Figure 1. Standard control chart

above the central value. These two attributes, ease of use and support for longitudinal studies, are central to the usefulness of control charts.

The regression control chart described by Mandel (1969) and Ryan (1989, p. 272) satisfies the ease-of-use criterion. The regression control chart, Figure 2, has the independent variable on the horizontal axis and the dependent variable on the vertical axis. The predicted value is plotted and the control interval is established, based either on a confidence interval for the mean response (Mandel, 1969) or on a prediction interval for a new observation (Ryan, 1989, p. 273). Details of the statistical rationale may be found in standard statistics texts (e.g. Ott, 1984). Either way, the user simply enters the chart on the observed value of the independent variable and plots there the observed value of the dependent variable.

1.3. Shortcomings

Regression based on more than one independent variable (multiple regression) does not allow simple two-dimensional plots. Further, because each predicted value may arise from an infinity of sets of values of the independent variables, both the confidence interval for the mean and the prediction interval for a new observation vary with the values of the independent variables. Therefore, the user must calculate a confidence interval or a prediction interval for each observation using the values of the independent variables for that obser-

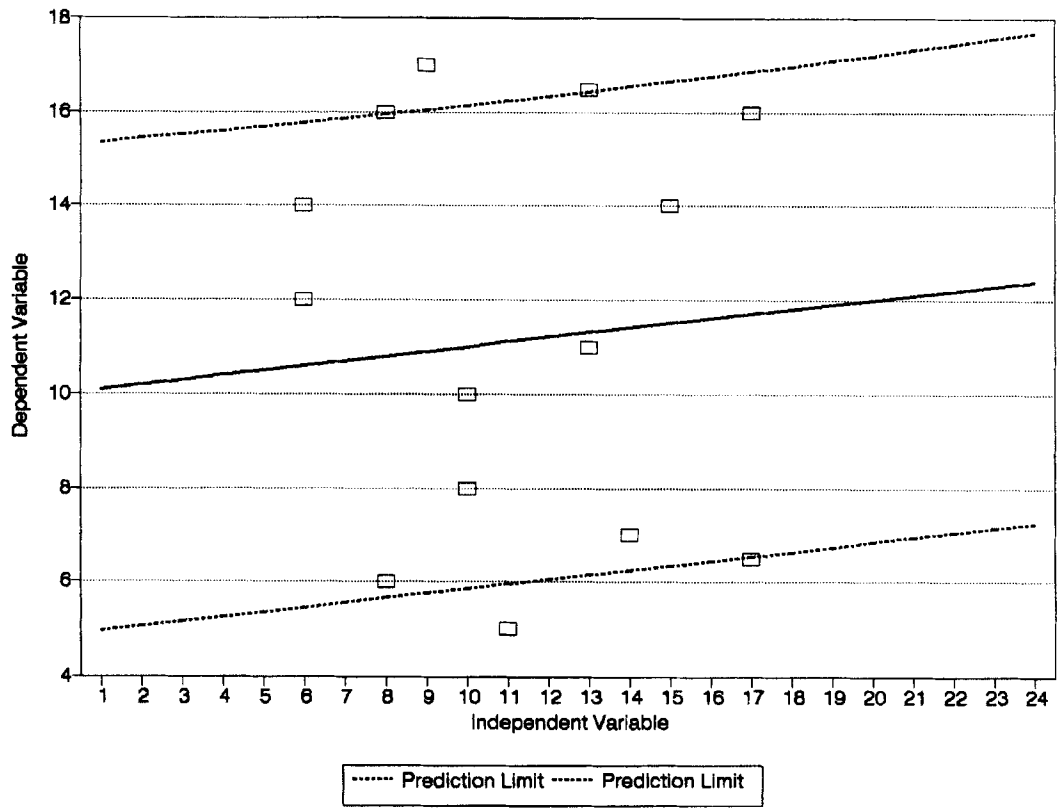

Figure 2. Regression control chart

vation. When performed manually, these calculations can be time-consuming, probably excessively so for most managers. Clearly, a multiple regression control chart has ease-of-use problems.

A second shortcoming is that regression control charts, as described by both Mandel (1969) and Ryan (1989), do not support a longitudinal view of activities. Rather than having time or sequence number or a similar attribute on the horizontal axis, the independent variable is plotted on the horizontal axis. The lack of sequence data in the plot denies the manager an important source of additional information about trends in the maintenance process. These trends may be observed in systematic deviations of the plotted points over time. This lack of longitudinal information reduces the motivation to expend the effort to prepare and use a regression control chart.

1.4. Approach

The answer to these shortcomings is to 'standardize' the deviation of the current observation from the predicted value and store it in time sequence. This may be done using the factor that is used to form a prediction interval, given by Ott (1984, p. 443) as

$$f_t = \text{SQRT(MSE)} \times \text{SQRT}(1 + (\mathbf{x}_o' (\mathbf{X}'\mathbf{X})^{-1} \mathbf{x}_o)) \quad (1)$$

The factor f_t is similar in concept to standard deviation. SQRT(MSE) is the square root of the mean squared error, available from the output of regression programs. $\mathbf{x}_o'$ is the row vector $[1\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$; $\mathbf{x}_o$ is a column vector of the same values; and $(\mathbf{X}'\mathbf{X})^{-1}$ is the inverse of the matrix of observations multiplied by the transpose of that matrix. $(\mathbf{X}'\mathbf{X})^{-1}$ is also available as output from regression programs. The difference between the observed value of the dependent variable and the predicted value for that variable, calculated from the regression coefficients and observed values of the independent variables, is divided by f_t calculated above. The result is a statistic that may be compared to the t-distribution for the desired significance level. Once 'standardized' in this way, the sequence of observations is comparable and meaningful in longitudinal evaluation.

The rest of the solution is to implement a user-friendly computer program to perform the necessary calculations and to do the plotting. Several possibilities exist. One is modern spreadsheet technology; another is an integrated statistical package such as SAS; a third possibility is a visual programming environment such as Visual BASIC. Programmed prompting and matrix arithmetic allow the calculation procedure to be made completely transparent. An important element in all three environments is the ability to generate graphs from the stored data, facilitating the interpretation of the results. This paper describes the development of a Visual BASIC application to generate a regression control chart used to assist the firm in managing software maintenance and identifying out-of-control situations.

2. APPLICATION DEVELOPMENT

2.1. Model

Having decided to try a multiple regression approach to process evaluation, the programming manager sought an appropriate regression model. Haworth, Sharpe and Hale (1992)

offer a conceptual model of software maintenance that had intuitive appeal to him. The model proposes that empirical investigations of software maintenance require control of four components: the environment, maintenance task, programmer skill and maintainability of the source code. Control in this context means that either the component be held constant throughout the study or that attributes of the component be measured so that its influence on the time to complete the maintenance process may be evaluated.

In developing a regression control chart for the firm, the environment was considered constant. The other factors (the maintenance task, the programmers' skill and the maintainability of the source code) were to be measured because they change with each maintenance task. It was decided that time, in programmer-hours, would be used as the dependent variable. Conceptually, the model became

$$\text{Time} = f(\text{Task, Skill, Maintainability}) \tag{2}$$

The details of the measurement of these factors are provided in Section 2.2, and the details of the regression model are described in Section 2.3.

The regression control chart may be used to identify instances of both exceptionally good and exceptionally poor performance. Presumably, good performance will be evaluated to determine what conditions should be repeated, the poor for conditions to be avoided. The programming manager understood that if an appropriate model could be developed, the effects of environmental changes might be observed in the plots of subsequent maintenance activities.

2.2. Measurement

2.2.1. Status

The measures used to form the regression model were selected because they appeared to satisfy the requirements of the conceptual model proposed by Haworth, Sharpe and Hale (1992) and because computer programs were available for two of the measures to facilitate data collection. Although exploratory in nature, published descriptions of these measures suggested that they would be useful in a regression model of software maintenance.

2.2.2. Time

Time was collected from the firm's task recording system. The task recording system is a computer-based system in which programming requests, task assignments, task status and time spent on the task are recorded. Company policy directs that the time recorded include time for analysis and coding, but not testing because of uncontrollable delays in getting test jobs run. The recorded figures were verified by interview to ensure the programmers measured their time consistently.

Time is recorded in programmer hours and varies from a minimum of 0.24 to a maximum of 42.00 (Table I). Drawn from programmers' activity reports across three months, the sample represents usual maintenance activity for the firm. In all, 53 separate maintenance tasks are represented in the data set that was used to establish the regression model.

Table 1. Descriptive statistics of the data standard

Variable	Mean	Deviation	Minimum	Maximum
TIME	5.3826	6.9203	0.2400	42.0000
CHANGE	72.7925	166.9072	1.0000	1117.0000
PE	0.5845	0.2412	0.2469	0.8823
MAINT	14.7200	4.2341	5.7906	20.8123

2.2.3. *Maintenance task*

The maintenance task was measured in lines of code changed, added and deleted. This value was collected by comparing the father and son versions of the source code. The type of maintenance was examined in the maintenance request log with a view towards recording the type of maintenance. However, most recorded maintenance activity involved more than one type and contained no information to indicate the proportion of time devoted to each type. Therefore, only a count of the lines of code added, changed and deleted was recorded.

The number of lines changed ranges from 1 to 1 117 (Table I). The sample of maintenance tasks used to form the firm's regression control chart ranges from single-line bug fixes to extensive enhancement maintenance.

2.2.4. *Programmers*

Programmer skill was evaluated using an objective test because subjective evaluation of programmer skill has been shown to be error-prone (Vessey, 1985). Any objective test of maintenance skill could be used; the maintenance programmer skill test described by Haworth (1990) was selected because it was available at no additional charge. Before the set of maintenance data was collected, the programming staff were evaluated. Analysis of the programmers' responses was performed using PC ALSCAL to scale each programmer against the expert benchmark described by Haworth. The results of the scaling are the weights of a given subject on the three characteristic dimensions of experts (see Haworth (1990) for details).

The probability of being an expert was calculated for each programmer using linear discriminant analysis. The technique of linear discriminant analysis uses two groups in known categories (experts and novices, in this case) and with measured attributes (weights on the dimensions derived by PC ALSCAL). The objective of discriminant analysis is to minimize the classification errors. The discriminant analysis derives an equation that uses the values of the attributes to produce a discriminant value (D). The discriminant value is then used to produce a probability for each observation of being an expert. The discriminant function used to evaluate the programming staff was

$$D = 6.5278 - 7.47467\ D1 + 3.81131\ D2 - 15.43847\ D3 \tag{3}$$

where $D1$, $D2$, and $D3$ are the weights of the programmer on dimensions defined by the experts' combined benchmark. $D1$ represents a data versus process dimension; $D2$ represents a sequence of activities dimension; $D3$ represents a debugging heuristic dimension (Haworth, 1990, p. 53–55).

The value of D is used to calculate the probability of the programmer being an expert according to

$$P(\text{Expert}) = 1/(1 + e^{D}) \tag{4}$$

The range of probabilities for the personnel tested was from a high of 0.8823 to a low of 0.2469 (Table I). These probabilities are used to control for variability in programmer skill.

2.2.5. *Source code*

The maintainability of the source code is a key attribute for the model described above. Several code metrics have been offered as predictors of maintainability (e.g. Gill and Kemerer, 1991; Woodward, Hennell, and Hedley, 1979). For the current implementation, it was found that Arthur's (1983) maintainability metric was a better predictor than the others available.

Source code was analysed using a static analyser described by Arthur (1983). The analyser counts key words and various syntactic features of COBOL source code. It uses the data to calculate various 'simple' descriptive metrics such as consistency, simplicity, modularity and structuredness, among others. These simple metrics are then aggregated to produce several complex metrics. The rationale for these metrics, both simple and complex, is developed over several chapters by Arthur (1983, p. 132–142, 165–179), but the association between the measured attributes and human difficulty in working with the code is intuitive, as it is for all code-based metrics. Of primary interest in this study is the maintainability metric (M), which is defined as consistency + simplicity + modularity (Arthur, 1983, p.268). If these three simple metrics are decomposed into their components, the formula for the maintainability metric becomes

$M =$ number of page ejects / eloc/10
+ number of blank lines / eloc/10
+ number of comment lines / eloc/10
+ number of comment lines / eloc/100
− number of decision statements / eloc/100
+ number of entry points / number of exit points
− eloc/100
− structuredness
+ number of explicit calls
+ intrinsic function count / eloc/100 (5)

where eloc stands for executable statements and structuredness (s) is calculated as

$$s = ((g + 5a) / \text{eloc}/100)^2 \tag{6}$$

in which g is the number of GO TO statements and a is the number of ALTER statements (Arthur, 1983, p. 267). This definition seems to be more of 'unstructuredness' becauses it increases with the density of GO TO statements and ALTER statements. As it is applied, this metric attempts to quantify the harmful practice identified by Dijkstra (1968), and an increase in GO TO density reduces the maintainability value for the code.

The maintainability formula above follows Arthur's calculation (Arthur, 1983, p. 268) but differs in two respects: structuredness is only subtracted once whereas the published code causes it to be subtracted twice, and decision density is subtracted from maintainability (as suggested by Arthur (1983, p. 169–170)) rather than added, as in the published code.

The range of maintainability for the programs used to establish the regression parameters is 5.791 to 20.812 (Table I). Generally, the code is unstructured and lacks comments; on the positive side, the code lacks ALTER statements. The sequential, monolithic programs are typical of pre-structured-era code.

2.3. Regression model

The data were analysed using linear regression. The regression model was

$$\text{TIME} = \text{INTERCEPT} + \text{CHANGE} + \text{PE} + \text{MAINT} \tag{7}$$

where TIME is the programmer-hours to accomplish the maintenance, CHANGE is the number of lines changed, added and deleted, PE is the probability of the programmer being an expert, and MAINT is the maintainability of the source code. The model is significant at $F = 89.919$ ($P = 0.0001$). The variance accounted for by the model (R^2) is 0.8463 of the total variance. All of the predictors are significant at the 0.05 level (Table II).

Plots of the residual values against the predicted value of time and against the values of the independent variables reveal no systematic violation of the regression assumptions. Several outliers exist, but because none of them can be explained, they remain in the data set. The statistical significance of the model is clear from the F statistic. The following discussion explores the practical significance of the model.

The INTERCEPT parameter estimate represents a base time for the maintenance activities sampled. This base time is defined as the time for a minimum change to code with a maintainability of zero without adjustment for the skill of the programmer. The basic maintenance event in this firm requires 10.44 programmer-hours.

Not surprisingly, the size of the CHANGE contributes to the time to make the change. The parameter indicates that each line of code that is changed adds 0.037 of an hour (2.2 minutes) to the time for the maintenance. Although statistically significant, not much can be done about the size of the change from a practical point of view. Fortunately, the contribution of each additional line of change is small (less than one per cent of the INTERCEPT value); in other words, a 10 line change will add only 22 minutes to a task that will take almost ten and a half hours.

The PE parameter estimate represents the contribution of programmer skill to the time

Table 2. Regression parameter estimates, parameter standard T for $H0$

Variable	Estimate	Error	Parameter = 0	Prob > $\|T\|$
INTERCEP	10.309469	1.75670945	5.941	0.0001
CHANGE	0.037121	0.00244214	15.200	0.0001
PE	−7.714099	1.64246459	−4.697	0.0001
MAINT	−0.220484	0.09430297	−2.338	0.0235

required for a maintenance task. The negative parameter estimate implies that a more skilled programmer will perform a given maintenance task in a shorter time. Assigning the most skilled programmer (PE = 0.88) to a basic maintenance event decreases the time by 6.79 programmer-hours; assigning the least skilled programmer (PE = 0.25) decreases the time by only 1.93 programmer-hours. Given the range of skill ratings in the shop, there is a 4.86 hour potential difference due to programmer skill.

The MAINT parameter estimate suggests that maintainable code reduces the time to make a change. Considering the range of values for the variable (5.791 to 20.812) and the size of the parameter estimate (–0.220), the effect of maintainability can be practically significant. At the low end of its range, maintainability reduces a basic maintenance event by 1.27 hour; at the high end of its range, maintainability reduces the event by 4.58 hour. From a practical standpoint, there is more than a three hour potential difference due to maintainability of the code.

The variance accounted for (R^2) by the firm's model is 0.8463 of total variance. Kerlinger and Pedhazur (1973, p. 437) state that values of R^2 near 0.66 are 'gratifying' in behavioural research. As a basis for a control chart, the firm's model meets Ryan's (1989, p. 270) suggested criteria of $R^2 > 0.80$. From these judgements, it was concluded that the calculated regression model was adequate for the purpose of constructing a control chart.

2.4. The control chart progam

The implementation of the control chart application began with specification of the input data. In addition to the dependent variable and the independent variables of the regression model, it was decided to collect the program identifier so that longitudinal studies of program maintainability can be made in the future.

The output variables were set at the predicted value for TIME and the t score for the actual value of time. The t score may be used to determine if the observation falls within the prediction interval by comparing it with the probabilities for the t distribution that may be found in most statistics texts (e.g. Ott, 1984).

Three control buttons were identified for the Visual BASIC form: Evaluate, Cancel and Graph. The Enter button invokes the Command1_Click subroutine that performs all of the calculations to evaluate the data entered on the form. The Cancel button returns control to an introductory form, and the Graph button invokes a plotting procedure that generates a line graph of the last 25 entries in the data file. Labels, text boxes for inputs, picture boxes for outputs, and control buttons were arranged on the Evaluate form (Figure 3).

The constants for the Evaluate form are shown in Figure 4. The constant values are from the regression model and are initialized when the form is first loaded. The values for the **XX** array are from the **X'X** inverse matrix which can be obtained as an option from SAS regression analysis. The values for Beta0, Beta1, Beta2 and Beta3 are the coefficients from the regression model, representing the intercept, and the coefficients for CHANGE, PE and MAINT respectively. The value for RootMSE is from the standard SAS regression report.

The subroutine Command1_Click calculates the predicted value of time using the Beta coefficients from the regression model (Figure 5). It performs the matrix arithmetic that is needed to calculate the factor for the prediction interval according to equation (1). The subroutine then calculates a standardized value for the difference between actual and predicted time and displays the results on the form.

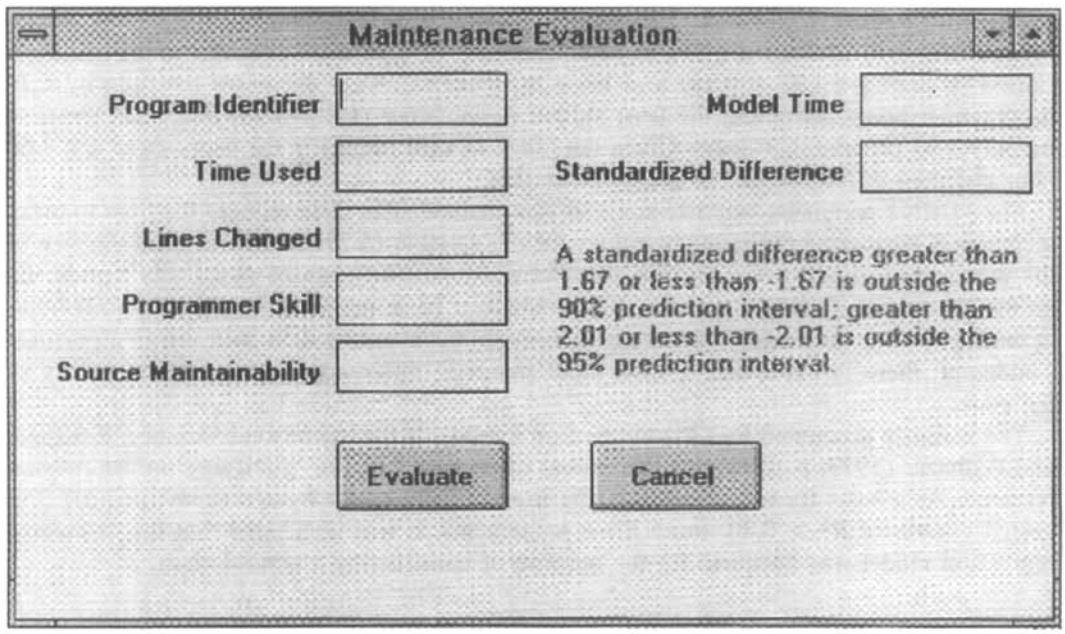

Figure 3. Primary screen

3. OPERATION

The user uses the report from the firm's maintenance task management system to determine the time spent on the task, runs a file comparison on the father and son versions of the program indicated in the report to determine the number of lines changed, and runs the father version of the program through the metric program to find the maintainability metric. In practice, it is expected that the programmer will run the file comparison and metric programs and include the results in the maintenance task report. With these data and programmer expertise ratings, the user is ready to enter the observation in the Evaluate Form (Figure 3). After the input data have been entered, the user clicks on the Evaluate button, and the results are displayed and recorded. One or many events may be evaluated and recorded during a session. At any point, the user may use the Cancel button to exit from the Evaluate form.

If the user wishes to see a graph of the most recent observations, including the one just evaluated, the user may click on the Graph button. This generates a line plot of up to 25 observations (Figure 6).

The plot in Figure 6 is of 18 observations made early in the use of this application. Clearly, observation 5 is off the scale. A subsequent interview with the programmer involved revealed that there had been an initial misunderstanding of the expected approach to the problem and that the programmer had attempted to reprogram an assembly language screen handler. After some time, additional information and guidance were given and the result was a 105 line change in the COBOL program. The total time, including the false start, was properly reported. Inadequate communication of task requirements was identified as an intermittant problem for this organization.

The significance of observation 5 was not noticed until it was evaluated with the regression model. The size of the change served to mask the fact that the 70 hours reported on the task

```
Rem Declaration of variables global to Evaluate.frm
Dim XX(1 to 4, 1 to 4) As Single
Dim RootMSE As Single
Dim Beta0 As Single, Beta1 As Single, Beta2 As Single
Dim Beta3 As Single
Dim in(1 to 4) As Single
Dim temp(1 to 4) As Single

Sub Form_Load ()
Rem  Initialize the X'X inverse matrix
  Let XX(1, 1) = .3950097881
  Let XX(1, 2) = -.000097096
  Let XX(1, 3) = -.195259617
  Let XX(1, 4) = -.01730857
  Let XX(2, 1) = -.000097096
  Let XX(2, 2) = 7.6339569 / 10 ^ 7
  Let XX(2, 3) = -.000103207
  Let XX(2, 4) = 6.9141672 / 10 ^ 6
  Let XX(3, 1) = -.195259617
  Let XX(3, 2) = -.000103207
  Let XX(3, 3) = .3453027324
  Let XX(3, 4) = .000065167
  Let XX(4, 1) = -.01730857
  Let XX(4, 2) = 6.9141672 / 10 ^ 6
  Let XX(4, 3) = .000065167
  Let XX(4, 4) = .0011383052
Rem  Initialize the regression coefficients
  Let Beta0 = 10.436763
  Let Beta1 = .037121
  Let Beta2 = -7.714099
  Let Beta3 = -.220484
Rem  Initialize the Root Mean Square Error constant
  Let RootMSE = 2.79509
End Sub
```

Figure 4. Global variables and constants

was excessive. It was not until after the calculation of the regression parameters that the programming manager had any sense of the amount of time that would normally be attributed to a 105 line of code change.

Observation 4 approaches the upper limit (95 per cent prediction interval) and is outside a 90 per cent prediction interval. A review of this observation found that the programmer had been assigned to do maintenance on an unfamiliar program. There were compelling business reasons for the assignment, but the identification of this task as being a borderline case reinforced the importance of assigning specific application systems to specific programmers for maintenance. Such a practice apparently allows the programmer to develop a familiarity with the programs of the system and to be more efficient at performing maintenance on those programs.

Other than observations 4 and 5, the plot shows no remarkable characteristics. There seems to be no systematic deviation such as that shown in Figure 1. However, the identification of one out-of-control task and another task on the borderline were sufficient to prompt the manager to undertake further refinements.

```
Sub Command1_Click ()
Rem  Clear output fields and declare local variables
  Picture1.Cls
  Picture2.Cls
  Dim i As Integer, j As Integer
  Dim out As Single, stdfactor As Single, pred As
Single
  Dim actual As Single, stdiff As Single
  For j = 1 To 4
    temp(j) = 0#
  Next j
  out = 0#
Rem  Translate string input to numeric variables
  Let actual = Val(Text2.Text)
  Let in(1) = 1#
  Let in(2) = Val(Text3.Text)
  Let in(3) = Val(Text4.Text)
  Let in(4) = Val(Text5.Text)
Rem  Compute a Predicted Value for the set of
independent values
  pred = Beta0 + in(2) * Beta1 + in(3) * Beta2 + in(4)
* Beta3
Rem  Compute the standardization factor
  For j = 1 To 4
    For i = 1 To 4
      temp(j) = temp(j) + in(i) * XX(i, j)
    Next i
  Next j
  For i = 1 To 4
    out = out + temp(i) * in(i)
  Next i
  out = out + 1#
  stdfactor = RootMSE * Sqr(out)
Rem  Standardize the difference
  stdiff = (actual - pred) / stdfactor
Rem  Output
  Picture1.Print Str$(pred)
  Picture2.Print Str$(stdiff)
End Sub
```

Figure 5. Calculation subroutine

A future development planned for the application is a pull-down menu of programmers and automatic insertion of the programmer expertise value. Work is underway to link the Visual BASIC application to the routine that calculates the program maintainability metric to make that element of data entry unnecessary. Finally, testing is underway on a maintenance planning routine that will produce estimates of programmer-time required based on the same regression parameters. This interest in extending and refining the control chart application is further testimony to the perceived promise that the application holds.

4. CONCLUSION

The regression model used in this implementation provides another case in which multiple components are needed to adequately model the software maintenance process. Further, the

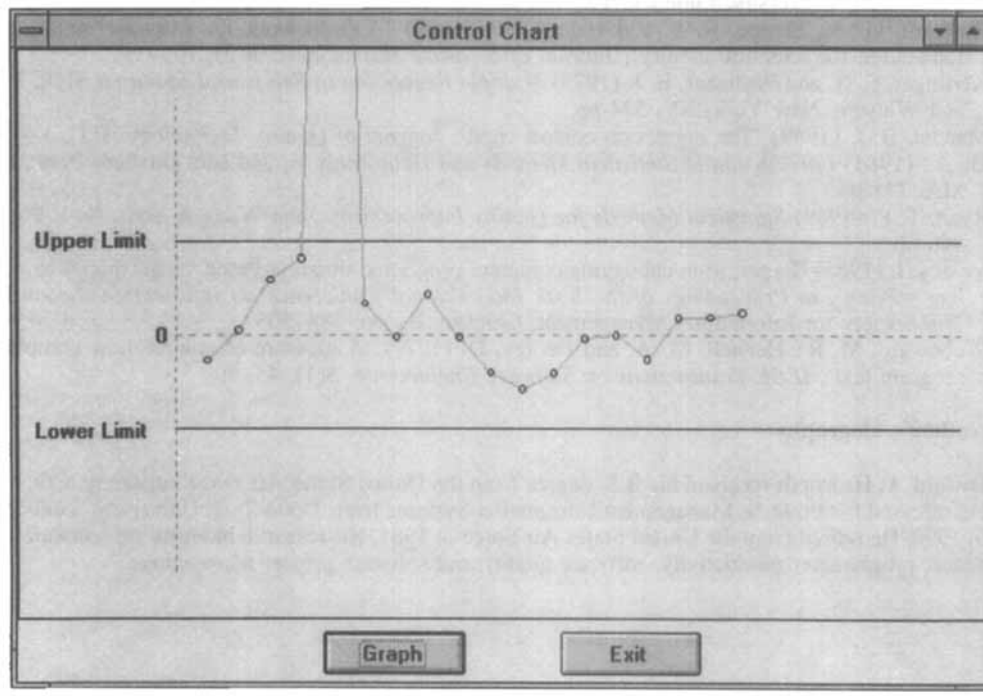

Figure 6. Regression control chart

R^2 value of 0.8463 supports the adequacy of the conceptual model articulated by Haworth, Sharpe and Hale (1992). Thus, this work lays a foundation for further research.

This implementation of the multiple regression control chart makes it as easy to use as the paper control chart. The implementation requires the same attention to measurement and perhaps slightly more effort in data entry. The result is delivered almost instantaneously. The user can see immediately whether the entry is within limits. Moreover, the record of past maintenance events can be displayed immediately and longitudinal inferences may be made.

The regression control chart makes obvious results that deviate from normal performance. Further, the development of the regression model produced information about the effects of inputs to the software maintenance process. Thus, all of the benefits of the standard control chart can be available to software maintenance managers who operate in a more complex and variable environment and need the capabilities of multiple regression to model that environment.

References

Arthur, L. J. (1983) *Programmer Productivity*, John Wiley & Sons, New York, NY, 288 pp.

Dijkstra, E. (1968) 'GO TO statement considered harmful', *Communications of the ACM*, **11**(3), 147–148.

Gill, G. K. and Kemerer, C. F. (1991) 'Cyclomatic complexity density and software maintenance productivity', *IEEE Transactions on Software Engineering*, **17**(12), 1284–1288.

Haworth, D. A. (1990) 'Software maintenance: measuring programmers' expertise', Ph.D. dissertation, Texas Tech University, Lubbock, TX, 122 pp.
Haworth, D. A., Sharpe, R. S. and Hale, D. P. (1992) 'A framework for software maintenance: a foundation for scientific inquiry', *Journal of Software Maintenance,* **4**(2), 105–117.
Kerlinger, F. N. and Pedhazur, E. J. (1973) *Multiple Regression in Behavioral Research*, Holt, Rinehart and Winston, New York, NY, 534 pp.
Mandel, B. J. (1969) 'The regression control chart', *Journal of Quality Technology,* **1**(1), 1–9.
Ott, L. (1984) *Introduction to Statistical Methods and Data Analysis*, 2nd edn. Duxbury Press, Boston, MA, 775 pp.
Ryan, T. P. (1989) *Statistical Methods for Quality Improvement*, John Wiley & Sons, New York, NY, 446 pp.
Vessey, I. (1985) 'Expertise in debugging computer programs: situation-based versus model-based problem solving', in *Proceedings of the Sixth International Conference on Information Systems, 1985*, The Society for Information Management, Chicago, IL, pp. 288–303.
Woodward, M. R., Hennell, M. A. and Hedley, D. (1979) 'A measure of control flow complexity in program text', *IEEE Transactions on Software Engineering,* **5**(1), 45–50.

Author's biography:

Dwight A. Haworth received his B.S. degree from the United States Air Force Academy, CO, in 1963. He received his Ph.D. in Management Information Systems from Texas Tech University, Lubbock, TX, in 1990. He retired from the United States Air Force in 1981. His research interests are software maintenance, programmer productivity, software quality and software project management.